Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Urh

Uporaba mobilnih senzorjev za identifikacijo kompleksnosti opravil

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Veljko Pejović

Ljubljana, 2016

University of Ljubljana
Faculty of Computer and Information Science

Gašper Urh

Mobile Sensing for Task Engagement Inference

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: doc. dr. Veljko Pejović

Ljubljana, 2016

Povzetek

Naslov: Uporaba mobilnih senzorjev za identifikacijo kompleksnosti opravil

Pametni telefoni so postali zelo zmogljive in osebne naprave, vendar v veliki meri ostajajo neizkoriščene. Samodejna identifikacija uporabnikove mentalne vključenosti v trenutno početje vse do danes še ni bila raziskana in bi lahko koristila na različnih področjih – od mobilnih aplikacij do sistemov za upravljanje s človeškimi viri. V tem magistrskem delu raziščemo možnost samodejne identifikacije zahtevnosti trenutnega uporabnikovega opravila z uporabo senzorjev v dostopnih pametnih telefonih. V ta namen razvijemo sistem za zbiranje podatkov, ki temelji na mobilni aplikaciji. Le-to javno objavimo in distribuiramo med uporabnike, zberemo podatke na strežniku in jih kasneje uporabimo pri metodah strojnega učenja. Najprej s pomočjo linearne regresije in nato še s klasifikacijo potrdimo obstoj šibke povezave med zajetimi podatki in kompleksnostjo uporabnikovega početja. Odkrijemo tudi, da so personalizirani modeli strojnega učenja bolj natančni od splošnih.

Ključne besede

pametni telefon, mobilno zaznavanje, strojno učenje, kompleksnost opravil

Abstract

Title: Mobile Sensing for Task Engagement Inference

Smartphones have become very powerful and personal devices, but still have to live up to their potential. To date, we have no automated means of uncovering a user's task engagement, which would be beneficial in numerous areas – from mobile applications to human resource management systems. In this thesis, we explore the possibility of automated task engagement inference using smartphone sensors. We try to find an answer by developing a data collection system based on a mobile application. We deploy and distribute the app among volunteers to collect data on our server. We then use machine learning approaches on collected data to uncover a weak link between task engagement and smartphone usage data and find out that the collected data is highly personalized.

Keywords

smartphone, mobile sensing, machine learning, task engagement

Acknowledgments

Foremost, I would like to thank my mentor, doc. dr. Veljko Pejović for his patience, useful discussions, remarks, engagement and immense knowledge through the process of researching and writing this thesis.

I would also like to thank all the volunteers that were willing to participate in the TaskyApp study. Without their contribution it would not be possible to conduct this research.

Last but not the least, I must express my gratitude to my family and my girlfriend for unconditionally providing me with support and motivation I needed at every step of my study.

Gašper Urh, 2016

Contents

Razširjen povzetek

Uporaba pametnih telefonov in z njo povezanih storitev v zadnjih letih skokovito narašča. Pametni telefon danes težko opredelimo le kot komunikacijsko napravo, saj nam poleg osnovnih možnosti klicanja in pošiljanja sporočil omogoča vrsto drugih uporab: vse od priporočanja restavracij in socialnega mreženja do navidezne resničnosti. Z nadaljnjo uveljavitvijo interneta stvari se bo število funkcionalnosti in mobilnih storitev le še povečevalo. Ker pa vsaka elektronska naprava zahteva uporabnikovo pozornost, je avtomatsko prepoznavanje uporabnikovih čustvenih in kognitivnih stanj zelo pomembno, saj bi se s tem lahko izognili nezaželenim učinkom – jezi in frustracijam.

Eno takih stanj je uporabnikova mentalna vključenost v početje. V kolikor bi bila naprava zmožna prepoznati kompleksnost trenutnega uporabnikovega opravila, bi se to odražalo v dodatnih funkcionalnostih pri obstoječih aplikacijah in razvoju povsem novih. Sporočilne aplikacije bi lahko zakasnile dostavo nepomembnih sporočil, kar bi uporabniku omogočalo ohraniti visok nivo koncentracije. Aplikacije, ki uporabnikom strežejo informacije, bi vsebino posodabljale le, ko bi uporabnik opravljal manj zahtevna opravila, kar bi lahko zmanjšalo porabo baterije in mobilnih podatkov. Razvili bi se lahko povsem novi sistemi za upravljanje s človeškimi viri, ki bi omogočali optimalnejšo porazdelitev dela med zaposlene in posledično znižali stroške podjetja, povišali delovno učinkovitost ter zadovoljstvo zaposlenih.

V sklopu tega magistrskega dela najprej opravimo pregled področja in ugotovimo, da je sorodno kognitivno stanje – uporabnikovo prekinljivost – mogoče zaznati tudi s pomočjo senzorjev vgrajenih v pametne telefone, pred-

vsem podatkov pospeškomera. Odkrijemo tudi, da je vpliv na kognitivno stanje odvisen od vsakega posameznika, na kar v veliki meri vpliva uporabnikova zmožnost večopravilnosti. To nas vodi do kognitivnega vpliva na izvedbo opravil in študijo, ki odkrije, da je kompleksnost opravila možno zaznati s pomočjo spremljanja velikosti zenice, kar zahteva uporabo posebnih naprav. Še vedno pa ni bila raziskana možnost zaznave zahtevnosti opravila s pomočjo dostopnih naprav. Na podlagi tega se odločimo za snovanje sistema, ki bi omogočil razkritje povezave med kompleksnostjo opravil in podatki pridobljenimi s senzorji dostopnih pametnih telefonov.

Najprej predstavimo snovanje predlaganega sistema za zbiranje podatkov, ki temelji na mobilni aplikaciji TaskyApp. Omejimo se na opravila znotraj pisarniškega okolja in definiramo merljivo kompleksnost opravila. Odločimo se za pet-stopenjsko Likertovo lestvico: "zelo lahko", "precej lahko", "niti lahko niti zahtevno", "precej zahtevno" in "zelo zahtevno". Zanašamo se na subjektivne ocene udeležencev raziskave, ki se dodajo posameznemu odčitku senzorjev. Na podlagi obstoječih del določimo tudi množico senzorjev, ki bi lahko vplivali na sklepanje zahtevnosti uporabnikovega početja: pospeškomer, žiroskop, Bluetooth, WiFi, lokacijski in časovni podatki ter nekateri drugi. Zaznavanje senzorjev mora biti zasnovano tako, da ne vpliva na delovanje operacijskega sistema in je čimbolj prijazno do porabe baterije. Nadalje določimo kdaj se posamezno zaznavanje sproži. Definiramo dve možnosti: ročni zagon zaznavanja s klikom na gumb in avtomatsko proženje ob zaznavi spremembe uporabnikove aktivnosti. Zaznani podatki se najprej shranijo na telefon, da jih ob slučaju ponovnega zagona sistema ne izgubimo, kasneje pa pošljejo na strežnik za uporabo pri postopkih podatkovnega rudarjenja in strojnega učenja.

Zavedamo se, da sama aplikacija ni dovolj za pridobivanje podatkov, temveč je zelo pomembno, da jo uporabniki redno uporabljajo. Z namenom aktivne uporabe aplikacije se poslužimo naslednjih prijemov: opominjanja na uporabo aplikacije preko sistemskih obvestil, uporabe metod igrifikacije, izdelave prijaznega in enostavnega uporabniškega vmesnika ter nagrade v

obliki 50 € kupona za enega od najbolj aktivnih uporabnikov.

Mobilna aplikacija je razvita v skladu s principi iterativnega razvoja, zato testiramo in analiziramo nove funkcionalnosti že tekom razvoja. Poleg tega se pred končnim zbiranjem podatkov odločimo za pilotno raziskavo, v kateri želimo celoten sistem testirati in izboljšati na podlagi mnenj uporabnikov. TaskyApp naložimo na dve napravi in po desetih dneh dobimo nekaj koristnih napotkov. Na podlagi mnenj in lastnih ugotovitev se odločimo za celovito prenovo obveščanja uporabnikov preko obvestil, popravimo nekatere dele uporabniškega vmesnika, odpravimo težave s strežniško implementacijo in omogočimo uporabnikom določitev časa, ki ga preživijo v pisarni. Ugotovimo namreč, da je veliko odčitkov zaznanih v času, ko uporabniki niso v pisarnah (npr. ob vikendih in pozno zvečer), zato želimo, da se avtomatsko zaznavanje v tem času izklopi. Podatki zajeti ob tem času bi nam onemogočali kvalitetno analizo podatkov v sklopu strojnega učenja, poleg tega pa bi z nepotrebnim zaznavanjem trošili baterijo.

Ko imamo dobro testirano in delujočo mobilno aplikacijo z želenimi funkcionalnostmi, jo distribuiramo desetim prostovoljcem. V namen lažje distribucije postavimo spletno stran s splošnimi napotki uporabe aplikacije in namenom naše raziskave, aplikacijo pa javno objavimo v trgovini Google Play. Po poteku pettedenske raziskave na našem strežniku zberemo skupno 3035 senzorskih odčitkov, od teh 232 z oznako kompleksnosti. Na podlagi opisnih statističnih podatkov ugotovimo, da dva uporabnika aplikacije nista uporabljala in da je večina, kar 82.3%, označenih odčitkov delo treh udeležencev. Nadalje ugotovimo, da je večina zbranih podatkov pridobljenih ob delavnikih med 8:00 in 17:00. K temu pretežno pripomore izklop avtomatskega zaznavanja aplikacije ob urah, ko uporabnik ni v pisarniškem okolju. Dnevna analiza zbranih podatkov kaže na to, da so bili uporabniki bolj aktivni v začetku tedna, medtem ko so oznake zahtevnosti pretežno konstantne od ponedeljka do petka. Nadalje analiza po urah pokaže, da je največ označenih podatkov zbranih med 10:00 in 11:00 ter da so poslane oznake rahlo zahtevnejše v popoldanskem času.

V naslednjem poglavju najprej opišemo postopke podatkovnega rudarjenja, s čimer iz pridobljenih podatkov izluščimo značilke. Le-te nam kasneje v postopkih strojnega učenja omogočijo pridobiti rezultate tega dela. Odločimo se za nekaj lahko izračunljivih značilk, kot so število Bluetooth in WiFi naprav v bližini, ura zaznavanja, pripeta oznaka težavnosti opravila in povprečne vrednosti surovih podatkov amplitude šuma in vseh treh osi pospeškomera ter žiroskopa. Podatke slednjih dveh uporabimo še za izračun povprečnih jakosti vseh treh osi, njenih varianc in števila prečenj srednje vrednosti. Zvočne signale in signale osi pospeškomera ter žiroskopa s hitro Fourierevo transformacijo preslikamo tudi v frekvenčno domeno in izračunamo spektre moči ter entropijo spektra. Rezultat podatkovnega rudarjenja je datoteka z vsemi izluščenimi značilkami.

Zadnji korak je modeliranje z algoritmi strojnega učenja. Najprej uporabimo linearno regresijo, ki nam razkrije podatkovno odvisnost od kompleksnosti opravila. V tem postopku določimo tudi značilke, ki najbolj vplivajo na končni rezultat. Izkaže se, da so premiki telefona, zaznani s pospeškomerom, vezani na lažja opravila. To je logična posledica tega, da smo se osredotočili na pisarniško okolje, kjer uporabniki večino delovnega časa sedijo, ob prostem času pa se verjetno premikajo. Na drugi strani se značilke žiroskopa in poznejša ura v dnevu odražajo v težjih opravilih. Prva verjetno zaradi rotacij naprave ob njeni uporabi, ko le-ta leži na mizi, medtem ko utrujenost uporabnikov ob koncu delavnika vpliva na zahtevnejšo percepcijo početja.

Izluščene značilke v postopku regresije uporabimo še pri klasifikaciji. Najprej vse naloge razvrstimo v dva razreda. Najlažji dve stopnji razvrstimo med "lahke" naloge ter vse ostale med "zahtevne". Tako razvrstimo 232 podatkov v 107 lahkih in 125 zahtevnih nalog. Uporabimo tri različne metode klasifikacije, kjer smo najbolj pozorni na natančnost napovedi in kritične napake. Kot kritične vrednotimo napake, pri katerih klasifikator razvrsti zahtevno nalogo kot lahko. Iz praktičnega vidika je taka napaka kritična, ker povzroči, da nam sporočilna aplikacija pošlje sporočilo v trenutku, ko smo zelo

zaposleni in nedovzetni do prekinitev. Kot referenčni klasifikator izberemo večinski klasifikator, ki je uspešen v 53,9%, in ga primerjamo z rezultati naivnega Bayesovega klasifikatorja in metode naključnih gozdov. Naivni Bayes se izkaže kot uspešnejši, saj je v 63,8% primerih natančen pri napovedi in ima hkrati razmeroma nizek odstotek kritičnih napak (13,8%).

Navade pri delu se med uporabniki lahko precej razlikujejo, zato se odločimo še za modeliranje podatkov uporabnika, ki izstopa po številu prispevanih označenih senzorskih odčitkov. Na podatkih uporabnika, ki prispeva 83 označenih odčitkov, uporabimo iste postopke, kot smo jih na skupnih podatkih, in dobimo natančnejše rezultate. Linearna regresija ponovno kaže na obstoj povezave med zbranimi podatki in zahtevnostjo opravil ter potrdi ugotovitev, da se rotiranje telefona in kasnejša ura odražata v zahtevnejših opravilih. V nasprotju s splošnim modelom se značilke pospeškomera tokrat odražajo v težjih opravilih. Razlog za to so lahko drugačne delovne navade ali drugačno pisarniško okolje tega uporabnika v primerjavi z ostalimi. Izluščene značilke ponovno klasificiramo v dva razreda in uporabimo večinski klasifikator kot referenco, ki je tokrat uspešen v 43 primerih (51,8%). Naivni Bayesov klasifikator se ponovno izkaže za bolj uporabnega od metode naključnih gozdov, saj je natančnejši pri napovedih (62,7%) in ima zelo nizek odstotek kritičnih napak (9%).

Magistrsko delo zaključimo z našimi predlogi za izboljšanje sklepanja o zahtevnosti opravil in pregledom omejitev dela. Osredotočili smo se le na pisarniško okolje in zato zbrali malo senzorskih odčitkov. Zato smo predpostavili, da so vsi odčitki označeni pravilno in zbrani v pisarniškem okolju (npr. niso bili zbrani v času vožnje s kolesom). Poleg označenih odčitkov smo v raziskavi zbrali večjo količino neoznačenih, ki bi jih lahko z algoritmi delno nadzorovanega učenja vključili v podatkovno analizo. Korak naprej pri sklepanju o zahtevnosti uporabniškega početja vidimo v uporabi nosljivih naprav, npr. pametnih zapestnic. Integracijo dveh takih zapestnic z mobilno aplikacijo TaskyApp smo že pričeli. Sledi identifikacija in diskusija o področjih, ki bi že lahko imela koristi od dobljenih rezultatov, ter pregle-

dom opravljenega dela. Glavna rezultata naloge sta razkritje povezave med uporabo pametnega telefona in kompleksnostjo uporabnikovih opravil, ter ugotovitev, da so personalizirani modeli podatkovnega učenja bolj natančni od splošnih.

Chapter 1

Introduction

Smartphones, nowadays, are very powerful and affordable devices that are normally kept close by their owners throughout the day, due to their pocket-size and usability. We can merely call them communication tools, providing us with a wide range of services: from online social networking, restaurant recommendations, tracking our exercise routine, over navigation in a new environment and immersing into virtual reality to name a few. The predictions show that close to a third of the whole world's population will be using one by the end of this year [18]. Further cohesion with other mobile devices into the Internet of things indicates that the reliance on mobile computing services is yet to grow.

In his 1991 manifesto, Mark Weiser outlined the need for the "stealth" ubiquitous computing device – the one that quietly blends with the lifestyle of its user [38]. We already depend on modern devices to notify us about important events, help us establish new friendships, take notes and other aspects that enable us to live interactive lives. However, our attention is required at times that are not suitable by beeping and flashing notifications. These unsuitable times can be very disruptive if a user is highly concentrated on a challenging task [28]. Thus, the user's current level of task engagement is of paramount importance in a number of ubiquitous computing scenarios. For example, a device knowing that a user is highly engaged in a task could

defer the delivery of an unimportant message, and notify the user only when the level of task engagement is lowered, thus reducing frustration and improving receptivity of the user to messages. On a bigger scale, inferring task engagement would be instrumental in human resource management systems, where it could help to equally distribute the workload among workers, resulting in stress reduction of the busiest workers, higher life quality and lower expenses of the company.

Up to recently, the inference of a user behavior has been outside of the scope of mobile computing. However, two factors, the increasingly personal use of devices and the ever-increasing sensing capabilities of devices, have opened up the opportunity for the automatic inference of certain aspects of human behavior, including mobility, physical activity, and even emotional state of a user. However, the automatic detection of task engagement, to the best of our knowledge has not been explored, yet. Thus, *the goal of our work is to explore the possibility of automated task engagement inference using commodity smartphones.* To fulfill the goal, we have to overcome the following challenges:

- Provide a measurable definition of task engagement levels.
- Collect sensor readings from users' mobile devices at moments pertaining to different task engagement levels.
- Label the collected data with the user-perceived task engagement levels.
- Apply machine learning algorithms to uncover a potential link between the sensed data and the task engagement quantifiers.

In this thesis, we tackle the above problems experimentally. We design and develop a smartphone sensing application that collects data from built-in sensors and, at the same time, interacts with the user to obtain the task engagement label – i.e. whether a user is engaged in an easy or a difficult task at the moment when sensors are read. We first review work related to our research and then focus on the app's designing process, implementation

of its main features, discuss overcome challenges and review our study, which we ran in order to collect sensor readings. In the study, we concentrate on the office setting and distribute the app to 10 users, who have collected a total of 232 data points. Next, we build machine learning models for task engagement, first using regression and then classification. In our findings, we show that there is an existing link between task engagement and smartphone's sensor data and present the most informative features. We manage to predict task engagement level with 63.8% accuracy (10% higher than our baseline). We then set a hypothesis that data tailored to an individual would yield more accurate predictions, and end up confirming it. Finally, we propose our guidelines for improved results on non-exploited collected data and identify additional sensors that could boost task engagement inference.

A part of the work presented in this thesis was published in a peer-reviewed paper presented at the UbitTention workshop in conjunction with ACM UbiComp 2016 [35]. This thesis, however, includes a more detailed description of the background and the experimental methodology we employed, and includes additional results obtained after the workshop paper has been published. Finally, in this thesis, we discuss further opportunities for automatic task engagement inference.

Chapter 2

Related work

In the interactive world where we are surrounded by many devices, each competing for a highly perishable commodity – user's attention [6] – human attention management is of great importance. Using "shadowing" technique González et al. showed that an information worker (analysts, software developers and managers) experience a high level of discontinuity in the execution of their activities [10]. They found out that an average worker spends three minutes working on any single event before switching to another. Further, the study showed that people spend on the average somewhat more than two minutes on any use of an electronic tool, application, or paper document before they switch to use another tool.

The above work indicates that multitasking has become a big part of our daily activities, hence very likely correlated with a user's cognitive involvement in a task. Salvucci and Taatgen proposed the idea of threaded cognition – an integrated theory of concurrent multitasking [32]. In threaded cognition, each task is represented with a cognitive thread. For example, writing an article and answering a phone call, one cognitive thread would be typing, and another operating the mobile phone. The theory provides explicit predictions of how multitasking behavior can result in interference for a given set of tasks. The perceived complexity of a task, which can be lowered through memory rehearsal, is critical for (concurrent) task performance [33]. It has

been shown that intermediate information, which is necessary for performing a task, is a bottleneck in multitasking [4]. In their study, Borst et al. define problem state resource, information that is directly and instantly accessible for the task at hand. While, on the other hand, it takes time to retrieve facts from declarative memory [2].

A cognitive state that determines the user's level of susceptibility to interruptions is interruptibility. Pejović et al. researched the relationship between task engagement and interruptibility using experience sampling method [28]. The study showed that although notifications allow users to defer interruptions for a later moment, the engagement with the current task still played a significant role in determining users' interruptibility. Some early works indicate that it is possible to detect interruptibility using dedicated sensors. In an office setting Horwitz et al. used camera and microphone to detect user's availability [12]. Fogarty et al. used "Wizard Of Oz" technique (a researcher – the "wizard" – analyzed long-term digital audio and video recordings of each participant's working environment) to figure out that interruptibility in an office setting could be detected using speech detector sensors [7]. Lilsys was developed as one of the first systems to detect interruptibility on-the-fly [3]. The system used ambient sensors (i.e. motion and sound), in order to infer certain cases of lower availability through machine interpretation.

However, in the meantime, smartphones came on the market with various built-in sensors. A novel machine-learning approach using smartphone sensors was proposed by Pielot et al. for predicting whether a user will see a notification message within the next few minutes [29]. In their two weeks study, data of 24 volunteers using Android mobile phone was collected and achieved 70.6% accuracy on predicting user's attentiveness using only seven easily-computed features (e.g. hour of day, screen status, volume settings). A smartphone library InterruptMe was used to recognize an opportune moment for interruption by training a personalized online classifier based on features of the accelerometer, location and time of day among others [26]. The classifier is updated on each user's feedback provided on a four-point Likert scale.

InterruptMe-based notifications result in shorter response times compared to randomly distributed notifications. Adamczyk et al. show that different interruption moments have different impacts on user's emotional state [1]. By predicting the best points for interruption, they consistently managed to produce less annoyance, frustration, and time pressure, required less mental effort, and were deemed by the user more respectful of their primary task. They suggest guidelines for an attention manager system which could enable a user to maintain a high level of awareness while mitigating the disruptive effects of interruptions.

The above works demonstrate that smartphones, with their sensing capabilities, are very practical and can be utilized to detect different cognitive states of a user. In another study, a high accuracy for detecting user's boredom was reached using Borapp mobile application [30]. The researchers collected data from 22 volunteers and developed machine learning models to automatically classify users in high or low boredom proneness with over 80% accuracy. Based on their findings it seems that a bored person is very likely less engaged in a task. O'Brien and Toms deconstructed the term engagement as it applies to peoples' experiences with technology [24]. They proposed a model of task engagement that focuses on the properties of a task that would compel more or less engagement, including the degree to which tasks are challenging, interactive, rich in feedback, aesthetically pleasing, enduring, and varied or novel.

Task engagement is challenging to infer, and to date, attempts have been made to infer task engagement by using physiometric sensors. Iqbal et al. used eye-tracker to predict task difficulty based on pupil dilation [20]. They show that a more difficult task demands longer processing time, induces higher subjective ratings of mental workload, and reliably evokes greater pupillary response at salient subtasks. However, to devise a practical, scalable task inference system, our goal is to explore whether commodity smartphones can be used for this purpose.

Chapter 3

Task engagement inference system

The possibility of detecting certain human cognitive states (e.g. interruptibility) with specialized equipment, using special techniques (e.g. shadowing technique, Wizard of Oz) and lately using a smartphone has been shown in the previous chapter. However, despite these advances, task engagement is yet to be reliably detected by commodity devices. Knowing task engagement would allow improved attention management systems and open a range of new possibilities for mobile apps. We decide to utilize powerful, personal and ubiquitous smartphones in order to build a system based on a data collection mobile application and back-end server for persistent data storage. In this chapter, we present the most important decisions in our data collection system design. We do so by defining a measurable definition of task engagement levels and providing the main features of our data collection app. The detailed implementation of the system is presented in Chapter 4.

3.1 Measurable definition of task engagement

A task is a rather broad term and for the purpose of our study we limit it to (mental) tasks performed in an office setting. Without such restriction,

we wouldn't be able to collect enough data points to extract meaningful and generalized results at the end. Furthermore, offices are rich environments in task dynamics (many different tasks with various difficulties) and office workers usually keep their smartphone close to them, or even use it, while working [39]. We define the following measurable five-level Likert scale (encoded in numeric values from 1 to 5, respectively): *"very easy"*, *"pretty easy"*, *"neither easy nor hard"*, *"pretty hard"* and *"very hard"*.

We are interested in the subjective experience of the task. Thus, we decide to rely on explicit task engagement labels provided by a user who answers a question about the perceived difficulty of the current task. There will be no default value, so the user will always have to choose a level of difficulty. We design TaskyApp, a mobile app that runs data sensing on background threads and enables users to provide a label for each sensing session.

3.2 Reading sensors

Next, we determine which data is most likely to be correlated with user's task engagement. According to the findings in the related work [3, 7, 26, 29, 30] and intuitively, we decide to obtain the following data per each sensing:

- *Accelerometer*: to detect phone movements
- *Gyroscope*: to detect phone rotations
- *Bluetooth*: whether enabled or not and the number of nearby devices
- *WiFi*: whether enabled or not and the number of access points available
- *Location*: longitude and latitude
- *Time*
- *Screen status*: capturing screen on and off events
- *Calendar events*: the number of active Google Calendar events

- *Sound*: ambient noise level
- *Ambient light level*
- *Phone volume settings*
- *Charging status*: the phone is being charged or not
- *Type of activity*: user's activity recognized by Google Activity Recognition API ("In Vehicle", "On Bicycle", "On Foot", "Walking", "Still", "Tilting", "Running" or "Unknown")

The sensing must be robust, crash-free and run periodically sampling in the background, which makes it difficult for efficient battery and system resources usage. One thing we must keep in mind is also building our system to be maintainable and easy to upgrade with new features – e.g. additional sensors.

3.2.1 Sensing strategies

Having a set of data to read from smartphones' sensors we need to design an approach how to initiate a sensing session. We come up with two different approaches, each having its pros and cons. **Sensing on demand** is done on user's request by a click of a button in our mobile application. The user is presented with a task engagement level chooser (with correspondent task engagement level descriptions) and an option to select a timeout before the start of a manually requested sensing session. This approach should get us more accurate task labels provided by the user, as she knows exactly what she will be doing at the time of the sensing. On the other hand, this approach will probably result in less labeled tasks and will require higher user's engagement.

With the other approach, **automatically initiated sensing**, we detect the user's context switch (e.g. the user picked his phone up from a desk) and initiate a new sensing session. We consider changes in user's activity (detected by Google Activity Recognition API) and location changes as suitable

indicators to detect a context switch. We also configure interval triggers (i.e. initiate sensing every half an hour) in order to be sure that we start adequate sensing sessions. The advantage of automatically initiated sensing approach is that we are guaranteed to get sufficient amount of sensor readings and equips us with an option to notify users to retroactively label detected tasks and engage them in using the app. However, as time passes since the sensing session, recall bias (users might not be able to correctly remember the correlated task engagement levels) should increase. Also, there is a possibility of a significant influence on draining the battery.

3.2.2 Long-term data storage

In our study, we greatly depend on the collected sensor readings and user-provided labels, so we cannot afford to lose any data points. Hence, we first introduce data caching in mobile app's local database, which ensures us that we do not lose data even if the user kills the app, restarts the operating system or the phone's battery gets drained. The cached data is later sent to our server via a WiFi connection in order to keep the user's mobile data plan untouched. On the server, data is persistently stored, so we can use it later for data mining and machine learning in order to find a possible correlation between the sensor readings and the user's task engagement.

For the fact that we are handling with sensible personal data, we decide to have our server, together with the database, located at our faculty. Each user has to agree to our terms of use at the very first launch of the mobile application, allowing us to send retrieved, anonymized data to our server. No matter when the user has an option to opt-out of our research and even delete all his data persistently stored on our server.

3.3 Engaging users

Since the main goal of the app is to collect labeled sensor readings we need active users to provide labels. Thus, we decide to make use of the following

four approaches for more frequent usage of the app:

- **Reminders via notifications**: we exploit notifications in a way to engage our users in using the app frequently and providing labels for detected tasks. The way of showing notifications reasonably changed after the pilot case study (Section 4.4). In case the user has not used the app for more than a day (and no other notifications were shown to her), we show her a simple notification. Clicking this notification opens the main screen of the mobile app with an option for sensing on demand. Alternative use of notifications is to inform the user right after an automatic sensing session concludes. Clicking on that notification redirects the user to a screen to provide a label.

- **Gamification**: positive effects of gamification have been shown by most of the empirical studies [11]. Therefore, we design a simple leaderboard, where each participant is compared relatively to others and gets a message (e.g. you are among top 20% of all participants). Further, we use user's collected data to show some statistics of her daily activities and plot her movements on a map.

- **Raffling a voucher**: as an incentive to provide the data and in order to boost the use of the gamification model, we decide to give away a 50 € voucher among active participants of the app (i.e. those who label the data regularly).

- **Intuitive user interface**: our priority is bringing the best possible user experience, therefore we keep user interface clean, simple and consistent with the design guidelines of the operating system.

Chapter 4

TaskyApp implementation

In this chapter, we present the implementation of our data collection system in detail. The programming code discussed in this chapter is available in our GitHub repository [37]. We decide to build the app on top of Android operating system. The system enables us to read the sensors we need, is reasonably easy for development and distribution, has a good programming API and, according to Statista, has the biggest market share, with a growing trend [17]. Since we need many system calls for the implementation of efficient sensing we develop a native Android application in Java programming language. Due to the support of low energy consumption standard – Bluetooth Low Energy – and distribution of over 70% in Android ecosystem [14], we decide to target our app to Android versions of 4.3 and above.

We reckon the data collection as the key phase in our research, thus we spend a significant amount of time designing and discussing key concepts of the app. We introduce a system architecture consisting of a mobile app for data collection and a server for centralized, persistent data storage. In Figure 1 we show all of the crucial architecture's components discussed in this chapter. User interface and user experience decisions are described at the beginning. Then, we talk more about TaskyApp's core features – sensing and data caching. At the end, we show app's communication with the server and persistent data storage.

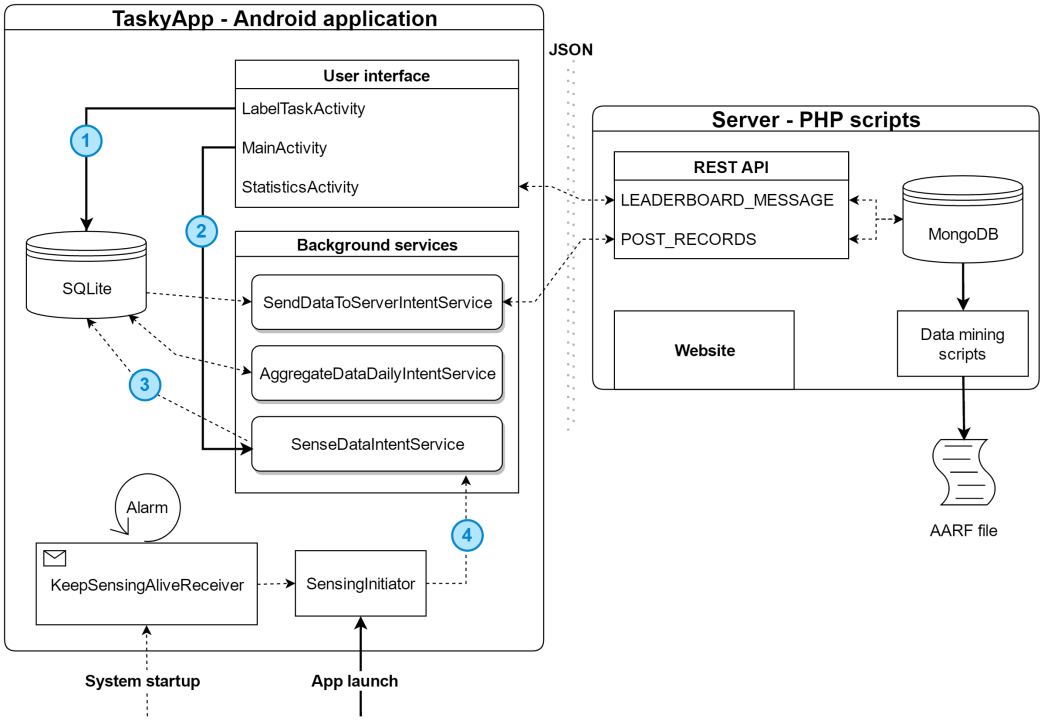

Figure 1: TaskyApp system's brief architecture diagram. The architecture shows the data flow in our system – from user interface interactions and sensing components in the app to persistent storage and data mining scripts on the server. User initiated actions are marked with solid connections, whereas dashed indicate actions instrumented by the system. The four most important connections are denoted with numbers: 1 = User provides a label for a task, 2 = User manually requests a new sensing session, 3 = New sensor data, 4 = User's context switch detection.

4.1 User interface

Designing the user interface (UI) proves to be difficult as some of our users do not have a deep knowledge of our task engagement study. Hence, we keep the app's UI minimalistic, with the background execution mostly abstract to the user. We decide to design our app according to Android's conventions, following the latest Material design guidelines [15], which makes the

UI familiar and understandable to our users. The UI should be positive and vibrant, therefore we, consistently throughout the app, use blue as a primary and yellow as a secondary (accent) color. For most of the texts, we use darker accents on white surfaces for good readability.

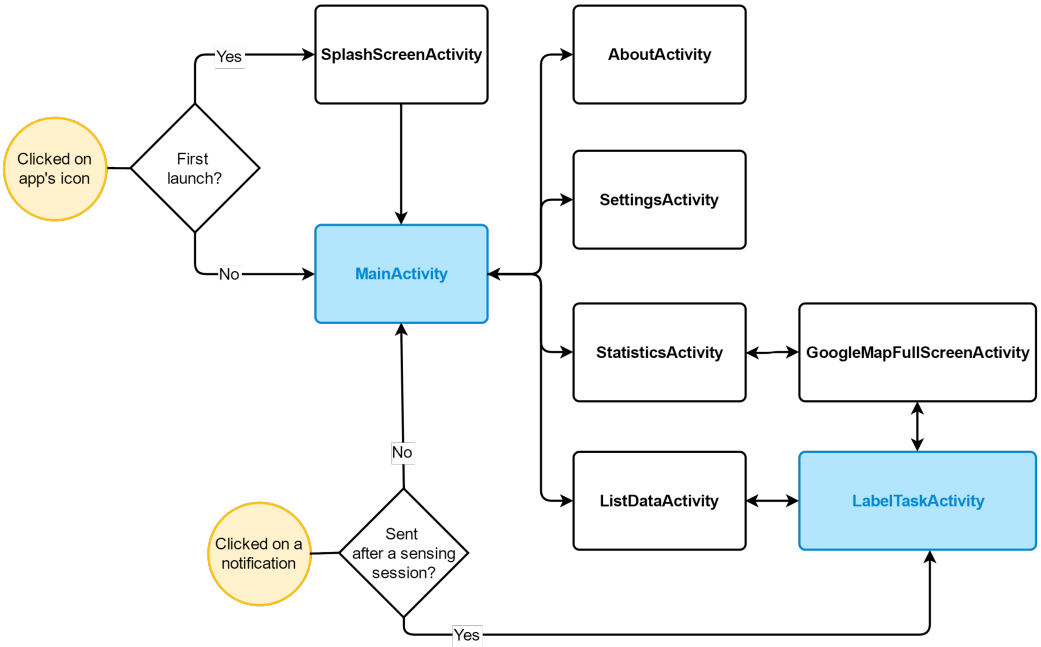

Figure 2: TaskyApp's user interface views. Rectangles indicate Android Activities, diamonds decisions and circles entry points to TaskyApp. The two most important views for data collection are colored in blue.

The UI of our app is mostly build of Android activities, shown in Figure 2. `Activity`[1] is an Android component that provides a screen through which users can interact with the application. Each `Activity` is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and floats on top of other win-

[1]We denote Android `Activity` with a different font to distinguish between the operating system's components and a user's activity.

dows [13]. TaskyApp is made of eight `Activities`, loosely bound to each other. Only two are essential for the fundamental purpose of the app (colored in blue), data collection. Other `Activities` deal with guiding a user through the app, providing help and managing application settings. The purpose of each `Activity` and transitions between them are described in the following paragraphs.

`MainActivity` is the main view of the application (Figure 3). It enables a user to manually start a new sensing session and has buttons to access other screens. `MainActivity` shows up every time (except for the initial launch) the app is opened via Android's application launcher. An essential part is an option for sensing on demand. We want to emphasize this option, thus we make the button for sensing on demand raised and others flat. The user can choose a difficulty on the Likert scale, using the provided slider, and the time when the task commences (e.g. "I am starting a pretty hard task in 15 seconds"). The explanation of the available Likert scale (Section 3.1) is available to the user in a pop-up window with a click of the button next to the slider. By clicking the "Start sensing" button the countdown starts (Figure 4) and the sensing invokes right after the given timeout. On completion of the sensing session, we inform the user and label the sensed data with the provided task engagement label.

`LabelTaskActivity` is, along with `MainActivity`, the core feature of TaskyApp's user interface. The user is provided with an option to label sensed task, using the same component as in `MainActivity`, or discard it (Figure 5). We again use a raised button to change user's focus towards labeling the task rather than discarding it. We provide information regarding that task as it is difficult to remember what you were doing at a certain time of a day. We show the task's location on a map, time of sensing and detected phone state (whether the phone was tilting, being still and other activities captured by Google Activity Recognition API). `LabelTaskActivity` is accessible either via `ListDataActivity` or directly via a notification shown to the user right after an automatic sensing.

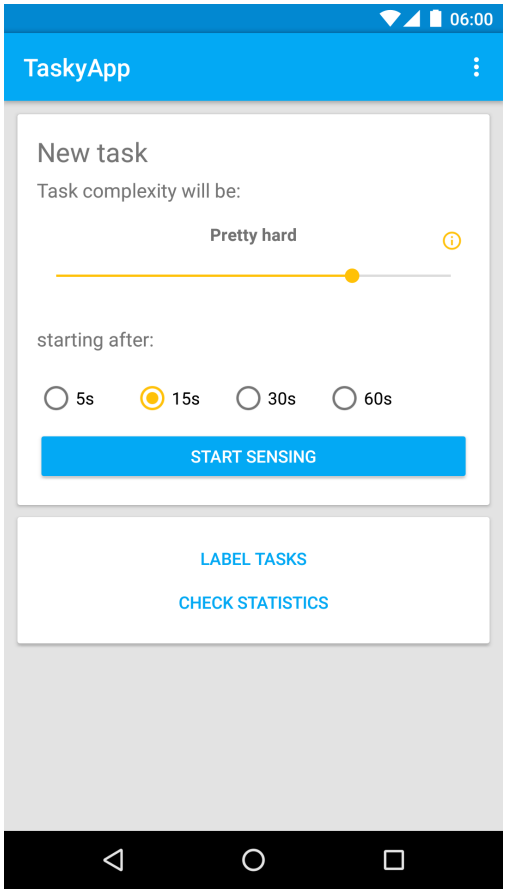

Figure 3: TaskyApp's main view – `MainActivity`

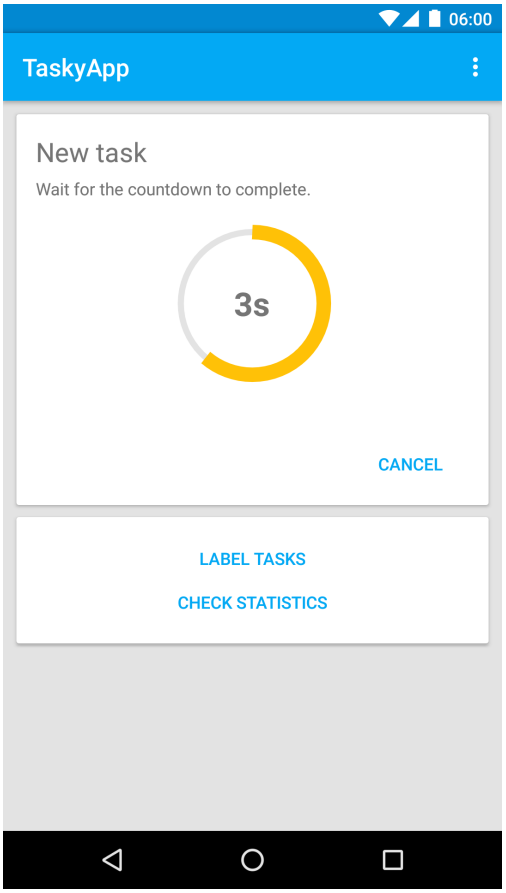

Figure 4: `MainActivity`'s view after click on "Start sensing" button

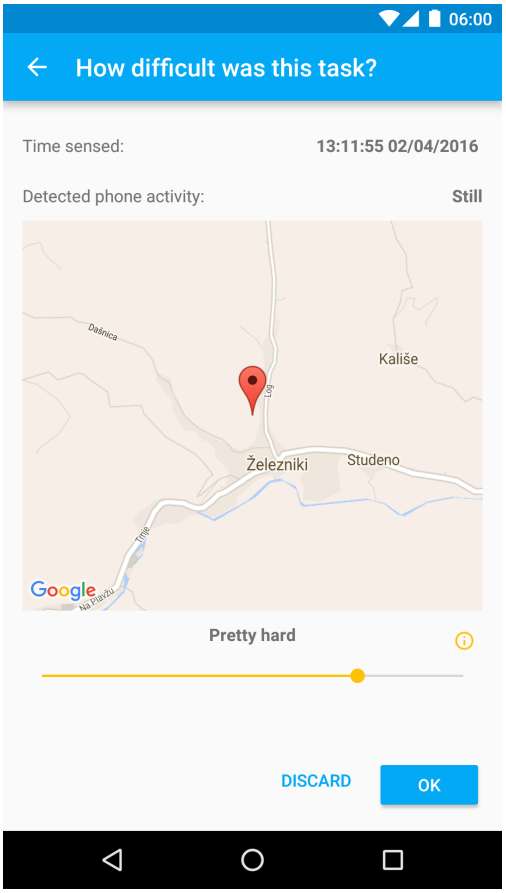

Figure 5: `LabelTaskActivity`

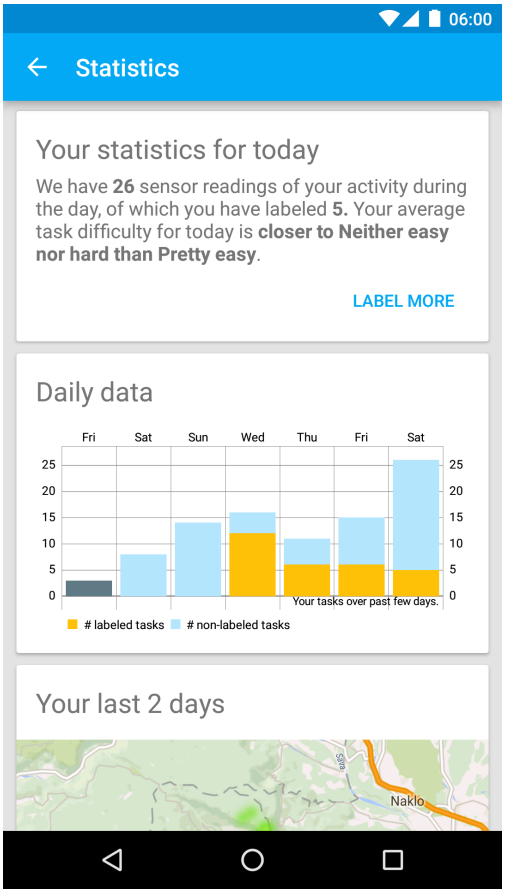

Figure 6: `StatisticsActivity`

ListDataActivity contains a simple list of up to ten, randomly selected, non-labeled sensed activities throughout the last two days. We decide to limit the number of tasks, so a user can feel that some progress has been made after labeling a task. By clicking any of the tasks, the user gets redirected to the `LabelTaskActivity`. `ListDataActivity` was developed with retroactive labeling in mind, which proved to be inefficient in the pilot case study, thus it is not frequently used. The `Activity` is accessible via `MainActivity`'s "Label tasks" button.

StatisticsActivity is developed to engage users with a simple gamification model (Figure 6). The user can see a heat map of his movements over the last two days, his daily statistics and aggregated data on a histogram since the beginning of the study. Besides, one can also check his progress relatively to other participants of our study on a leaderboard. The leaderboard component is completely configurable from the server, i.e. we can modify the title and the message or completely hide that component.

SplashScreenActivity is the first screen shown to the user after the app's installation. The user is provided with instructions on how to use the app, a short description of the research, what the app is about, an option to select his office hours and to provide an optional contact email. Moreover, the purpose of the study and terms of use are presented, to which each user has to agree.

GoogleMapFullScreenActivity is a simple view containing only a map and options to show a heatmap of user's movements or absolute locations as pins. Clicking a pin provides details about that task. If the task has not been labeled yet, the user has an option to provide one by using `LabelTaskActivity`.

SettingsActivity has some important features and equips a user with options to modify details provided in `SplashScreenActivity`, to change notifications settings and opt-out of the study. **AboutActivity** serves only static information about the authors and a link to the app's website [36].

Apart from the conventional Android app's launch, we also provide options to launch the app via notifications. Clicking on a notification to remind users of TaskyApp usage opens the `MainActivity`. The other, shown right after an automatic sensing, opens `LabelTaskActivity` to provide a label.

4.2 Background sensing

TaskyApp's preeminent functionality is data collection – sensor readings and user labels. In the previous section, we discuss how we attempt to get users' labels, while in this section we focus on how TaskyApp handles getting sensor readings and related challenges we tackle. There are few requirements we need to consider: the app must work fast, be battery efficient and sense data seamlessly – the app should not interrupt other running apps or block the system. Hence, we introduce sensing running simultaneously on several background threads (Figure 7). This persuades us to build a loosely bound sensing component, hence we take advantage of `IntentService` class in Android. The class handles asynchronous requests on demand in its main method, `onHandleIntent`, which runs on a worker thread and stops itself when it runs out of work. We extend it to `SenseDataIntentService` class (Figure 1) to handle all the sensing, independently of other components, in TaskyApp.

The sensing method is called on every manual sensing request and on automatically detected context switches. We also configure an alarm to automatically call the method every half an hour, in case no context switches are detected. Once called, if the sensing session has been initiated automatically, we first decide whether the time is appropriate to start sensing, if not, we stop the initiated sensing session to preserve battery and system resources. All of the following conditions must apply, listed by importance:

1. Right now are office hours

2. No less than 10 minutes have passed since the last sensing session.

3. The user's activity, detected by Google Activity Recognition API, has changed or location changed for more than 35 meters (both values, if available, are checked at the start of each sensing session)

Next, sensing shown in Figure 7 initiates. It is vital that we get various sensor readings concurrently, thus we implement `SensorThreadsManager` class to take care of thread management. It exploits two Java classes, `ExecutorService` and `CompletionService`, for parallel thread execution. Its main methods are `submit` and `take`. The first submits a task (a `Callable` object) for execution, whereas `take` retrieves and removes the next completed task, waiting if none are yet present.

In order to read sensors efficiently, we take advantage of the open-source third-party library and its pivotal class `ESSensorManager`. The main goal of the library is to make accessing and polling for Android smartphone sensor data easy, highly configurable, and battery-friendly [21]. We configure `ESSensorManager` to sense accelerometer, gyroscope and microphone (colored in gray) in a ten seconds long window, while Bluetooth and WiFi (colored in yellow gradient) sensors stay turned on only until we get all nearby devices – to save battery, as this is usually less than ten seconds. Next, we

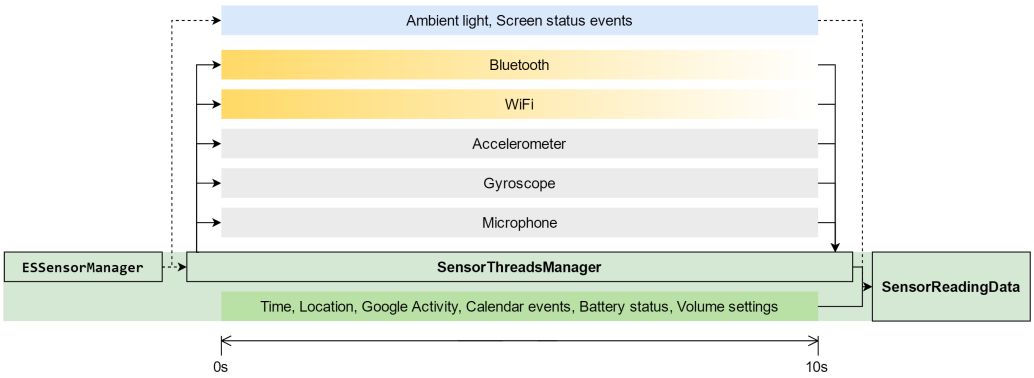

Figure 7: Parallel execution of data collection in TaskyApp. Each row represents a process thread. Colored in green is the main sensing thread which invokes other threads for parallel data retrieval.

create a `Callable` object for each of those five sensors and submit it to `SensorThreadManager`. The sensors are sensed immediately and simultaneously.

All mentioned sensors in the previous paragraph are of type pull – turned on only if an application requests so. Other available are push, Android broadcasts changes to all applications, and environmental sensors. We subscribe to screen status (push) and ambient light sensor (environmental) events, both are colored in blue. We do so by using `ESSensorManager` at the beginning of the ten-second sensing window to listen for value changes and unsubscribe after that window ends.

Apart from sensors discussed so far, we also retrieve other data directly using operating system APIs (colored in green): time, location, Google Activity, active Google Calendar events, charging status and volume settings. That data is captured at the beginning of the ten-second sensing window. All of the captured data is then stored in a single object – `SensorReadingData`. We also add additional informative fields of the sensing session: time of completion, start time in a readable form, sensing policy (how the sensing was initiated) and the app's version. We wrap the object in a new `SensorReadingRecord` object and cache it in our SQLite database. At this point we broadcast an Android `Intent` object, notifying other components (e.g. `MainActivity` and `ListDataActivity`) that sensing has just finished.

4.2.1 Robust sensing

One important feature of TaskyApp is also keeping sensing alive even after the operating system reboots or the user kills the app. The implementation of this functionality is presented in the bottom left corner of system's architecture diagram (Figure 1). `KeepSensingAliveReceiver` extends `BroadcastReceiver` class and keeps the sensing components up and running. We configure its `IntentFilter` to listen for system boot events and custom defined action "KeepAliveAction". Apart from receiving oper-

ating system's start-up events we also configure an alarm, which uses the custom action to wake up the device every half an hour. On each call we check the status of sensing and rerun it, in case it is stopped, by utilizing `SensingInitiator` class. The class sets up context switch detection by subscribing to location and Google Activity Recognition API changes. It also starts an interval alarm, ensuring us to get at least one sensor reading approximately every half an hour. `KeepSensingAliveReceiver` gets called also on every app's launch.

4.3 Persistent data storage

All sensed data is cached in the app's local database for at least two days. During that period, we endeavor users to retroactively label the data. Afterward, the data is either sent to our server for persistent storage or discarded. To that end, we implement another `IntentService`, which is called (at least) once a day. First, we check if there is any available cached data older than two days to run our data aggregation method. The average label, number of all and number of labeled readings are derived out of each day's collected data and saved into `DailyAggregatedData` object. The object is stored in the local database and used later to build the histogram in `StatisticsActivity` (Figure 6).

Subsequently, we try to send available data to our server for persistent data storage. Therefore, we first check for WiFi availability, in order to preserve the user's mobile data plan. In case the device is connected to a WiFi connection, we query the local database for all sensor readings older than two days, otherwise quit the `IntentService` and listen for WiFi connectivity changes to call the service once again. We then select all labeled and eight randomly selected non-labeled tasks per day (the decision is discussed in Section 4.4) and do an HTTP POST request to one of our REST API endpoints at our server (Figure 1). We put a raw JSON in the POST header, which is of the same structure (only without the "_id" attribute) as shown in

Listing 4.1. The server responds with another JSON, confirming all successfully stored sensor readings by sending an array of integers, "database_id"s. On response, TaskyApp deletes all records with confirmed ids from the local database.

4.3.1 Server-side implementation

An empty virtual machine on the faculty's VMware vCenter server has been allocated to us. We decide to implement our server-side programming logic in PHP scripting language and exploit NoSQL database management system, MongoDB. In our case both provide us with a simplicity of use, adequate performance and efficient manipulation with data in the JSON format. Considering that, we install Apache server on Ubuntu operating system, PHP, MongoDB and set up FTP and SSH for remote access.

In total, we differentiate between three REST API calls (Table 1). All of them are available through the same URL, where we route each call based on a GET parameter named "action". HTTP POST method is used for all calls since we have to identify the request's source device. Every request sends a JSON payload in the HTTP header of the same structure as presented in Listing 4.1. The payload always contains an authentication object (attribute "auth"), consisting of a device id along with an optional contact email, and related data (attribute "data").

	action	**method**	**attributes**
Post records	post_records	POST	auth, data
Opt out	opt_out	POST	auth
Leaderboard message	leaderboard_message	POST	auth

Table 1: List of all server REST API endpoints. In the "action" column we list values of a GET parameter used for routing to the desired functionality. The "attributes" column indicates at the JSON structure, while "method" denotes the type of endpoint's HTTP request method.

```
{
        "_id": ObjectId("570ffd5ebe4c7371c6357aef"),
        "auth": {
                "device_id": "309a3c10d19a8b3a",
                "email": "anonymous@server.si"
        },
        "data": [{
                "accelerometer": {
                        "meanX": -0.49076846,
                        "meanY": 0.6094682,
                        "meanZ": 9.4937525,
                        "values": [
                                [-0.50315857, 0.51475525, 9.440842],
                                ...
                        ]
                },
                "activity": {
                        "type": "Still",
                        "confidence": 100
                },
                "app_version": 7,
                "database_id": 3,
                "environment": {
                        "ambient_light": {
                                "max": 27,
                                "max_range": 10000,
                                "mean": 25.61514,
                                "min": 24
                        },
                        "bluetooth_turned_on": true,
                        "battery_charging": false,
                        "num_bluetooth_devices_nearby": 5,
                        "num_wifi_devices_nearby": 0,
                        "wifi_turned_on": false
                },
                "gyroscope": {
                        "meanX": 0.002849017,
                        "meanY": -0.0025972922,
                        "meanZ": -0.0016012477,
                        "values": [
                                [0.0030975342, 0.00012207031, 0.00062561035],
                                ...
                        ]
                },
                "label": 2,
                "location": {
                        "accuracy": 49,
                        "altitude": 339,
                        "lat": 46.0536117,
                        "lng": 14.5196431
                },
                "microphone": {
                        "amplitudes": [3903, ...],
                        "max_amplitude": 9994,
                        "mean_amplitude": 2917.9191919191917,
                        "min_amplitude": 0
                },
                "screen_status_list": [],
                "sensing_policy": "USER_FORCED",
                "t_ended": "1461052528246",
                "t_started": "1461052517940",
                "t_started_pretty": "09:55:17 19/04/2016 "
        }]
}
```

Listing 4.1: **Structure of a document in MongoDB for each user.** This example shows one user's sensor reading data. Post records API's call payload has the same structure, except for the MongoDB specific _id field. The ellipses at the end of each array indicate more values.

We design a very simple database data model, which enables us to easily store sensor readings in form as we get them. MongoDB is an open-source document database that provides high performance, high availability and automatic scaling. A record in MongoDB is a document, which is a data structure composed of field and value pairs [16]. We keep the same structure to the JSON payload sent via the app – except for automatically created identification field "_id" (Listing 4.1).

Post records is the main endpoint of our REST API. It permanently stores sensor readings sent from the mobile application. On each request, we validate received payload and use "auth" JSON field to check if the particular user already exists in our database. We do so by checking the "device_id" value. If the user does not exist we create a new document with the same content as the received payload, otherwise we merge values in "data" array with existing values in the MongoDB document (omitting possible duplicates). Besides, we always update "auth" field in the database to be identical to the one received. This comes handy when a user changes his contact email in TaskyApp's settings.

Other two implemented endpoints are less frequently used, but still important for TaskyApp's functionalities. **Leaderboard message** API call runs our simple gamification model and provides a simple message that reports to the user how many labeled sensor readings has she provided proportionately to the other participants in the research. If the user provides enough labels to be among top 20% of the study's participants, this call would generate the following message:

> "Well done! You are among 20% of all TaskyApp users. Your chances of winning the voucher are very high, keep up the good work."

Apart from the message we also send an optional title and an attribute that hides the leaderboard component in TaskyApp if set to false. The user is again identified by device id found in "auth" field of the received payload. In the same way, we use the identification process in the **opt-out call**. It

is a simple call that removes all of the user's content, all of his MongoDB documents, stored in the server's database.

4.4 Pilot case study

We develop TaskyApp using an iterative approach, where we test and analyze its functionalities throughout the development process. After the app was developed with functions we considered to be important, we ran a small, preliminary study, to test the system for bugs and to improve the app's user experience. Since we want to collect quality data, distribute a crash-free app and engage users into actively using the app both are of particular importance. For crash reporting, we take advantage of Crashlytics, available in the Twitter's Fabric suite [19].

We installed TaskyApp on two mobile phones and kept it running for ten days. During that period, we noticed several bugs and UI glitches in the app. We fixed most of the crashes detected by Crashlytics and thoroughly tested the server's REST API implementation and persistent data storage.

In the first version of the app, we did not include the office hours option, so automatic sensing took place throughout the day, also on weekends. That resulted in a lot of inappropriate sensor readings for our purpose of recognizing task engagement in an office setting (e.g. a sensing session took place while jogging), making it difficult to learn from the data. In addition, that caused higher battery consumption than necessary. Consequently, we have provided users with an option to select office hours, defaulting from 8:00 to 16:00, and an option to exclude weekends.

More importantly, we noticed that it is very easy to forget about the app and not provide much needed task labels. Back then, we relied on retroactive labeling. We sent two notifications per day – one sent at midday and the other in the evening to remind users of task labeling. That proved to be inefficient since it is difficult to recall in the evening which task exactly you were doing several hours ago, and resulted in incorrect labels and less

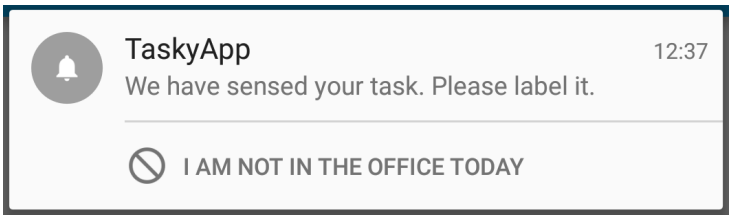

Figure 8: TaskyApp notification. A notification we send after an automatically initiated sensing session, the message's content changes between notifications to make it less monotonous. At the bottom, a user can find an option to stop sensing if she is not in the office that day.

labeled tasks. Hence, we have decided to notify users more actively and to send a notification right after an execution of automatically initiated sensing. Recurrence of those notifications can be changed in the settings and defaults to three per day. We introduce a simple algorithm that randomly distributes these notifications during one's office hours – meaning that notifications will not be sent at the same time each day. That proves to be more efficient, as users are reminded immediately, resulting in more accurate labels. In case the user is not in the office that day, we provide an action button embedded in the notification that stops sensing and removes all non-labeled tasks for that day (Figure 8). The user can disable such notifications, but will still get one notification per day, during her office hours, not to forget about the app.

Apart from mentioned glitches, we have also fixed user interface based on the received users' feedback, resulting in a nicer and smoother UI for manual sensing, cleaner statistics screen and more consistent UI across the app. We even identified some data, which could prove to be effective in inferring user's task engagement – phone volume settings and the user's active calendar events at the time of sensing.

Server's implementation fairly quickly started failing, which was detected by checking the server's log files. Sending too many non-labeled records resulted in exceeding MongoDB document's size limit of 16MB for a user, hence not saving sent records. As all records, until the limit hit were saved, but

not confirmed in the response, the mobile app did not delete them from the local database, thus sending them again in the next request. That resulted in several duplicated records in the server's database. We then make the server's implementation more fail-safe, with checks for duplicate entries. As a consequence, before sending to the server, we randomly select only eight (which is approximately 25% of all daily sensor readings on average) non-labeled data points per day, delete the others and send the chosen to the server. Duplicated entries in the database are later filtered out in the feature extraction.

Chapter 5

Data collection

Having a working and thoroughly tested mobile application, our next step is making it publicly accessible and to distribute it to end users. In this chapter we discuss the distribution of TaskyApp, running a study and show descriptive statistics of collected data.

5.1 TaskyApp's distribution

First, we develop and deploy a website [36] with the aim of helping with distribution and advertisement of the app (Figure 9). On the website, we first explain what the app is about, why it is worth installing it and a link for downloading the app. Next, we provide short instructions, backed with the app's screenshots, on how to exploit the app's main features. At the end, we show the consent form explaining the purpose of the study (the same as on the initial launch of TaskyApp) and contact details. The website is hosted on the same server as the REST API and is developed using Bootstrap framework, responsive – mobile first – design, because it is linked in the app and very possibly accessed via a mobile phone.

We make TaskyApp available through Google Play [34], the official app store for the Android operating system (Figure 10). First, we create an application entry in the store, enabling us to get a signing key needed for

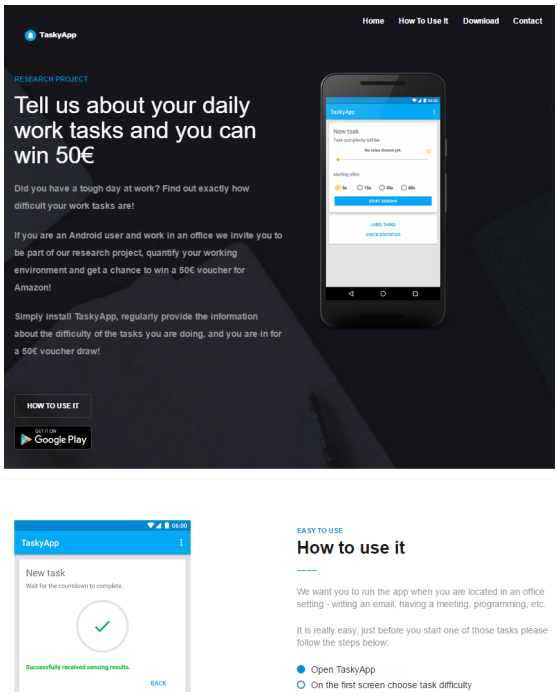

Figure 9: TaskyApp's website.

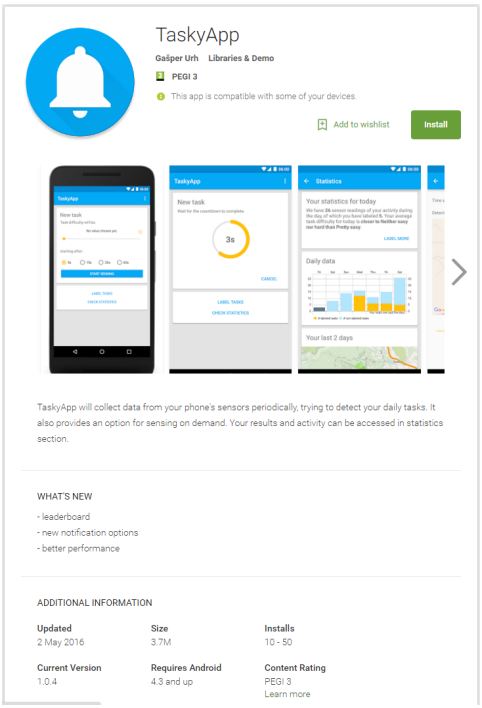

Figure 10: Google Play store listing of TaskyApp.

building and publishing the app. We then edit store listing information, upload promotional graphics and the signed APK to Google Play.

The crucial step was to find volunteers working in office settings and install the app on their mobile phones. We choose to distribute the app in person, which give us an option to further explain instructions on how to use the app properly, resulting in more quality sensor readings. That step proved to be more difficult than expected. We attempt to recruit participants through personal contacts. However, our potential users' concerns about the application's impact on the phone's battery life, the use of alternative smartphone platforms (e.g. iOS), and employment in professions that are not exclusively tied to an office setting prevented a wider distribution of TaskyApp through such a direct recruitment method. At the end, we distribute TaskyApp to ten different users (devices). Participants were from 23 to 56 years of age, four females and six males.

5.2 Conducting the full-scale study

After the app's distribution, we run a full-scale study for five weeks and collect data points from eight different devices (two users uninstall the app or decide to opt-out of the research in the first two days). We have kept the app running on some phones even after, so we got some additional data after that period.

User	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Total**
Num. of labels	83	57	51	15	11	8	4	3	232
Average label	2.51	2.68	3.47	1.93	2.55	3.38	2.25	1.67	2.74

Table 2: Per user task label distribution. Tasks are labeled with numeric values from 1 ("very easy") to 5 ("very hard").

In total, we collect 3035 unique sensor readings stored on our server, of which 232 include task difficulty labels. We show labeled task distribution per

user in Table 2. Since users' data is anonymized in our study, we use digits from 1 to 8 to identify users in the table, sorted by the number of labeled tasks provided. We can see that most of the labeled tasks were provided by three users – 191 (82.3%). On average the task complexity is just 0.26 short of the medium label – "Neither easy nor hard". Most of the tasks are labeled as "Pretty easy" (31,9%), followed by 24,6% and 24,1%, "Neither easy nor hard" and "Pretty hard", respectively. We are short of tasks labeled as "very easy" (14,2%) and "very hard" (5.2%).

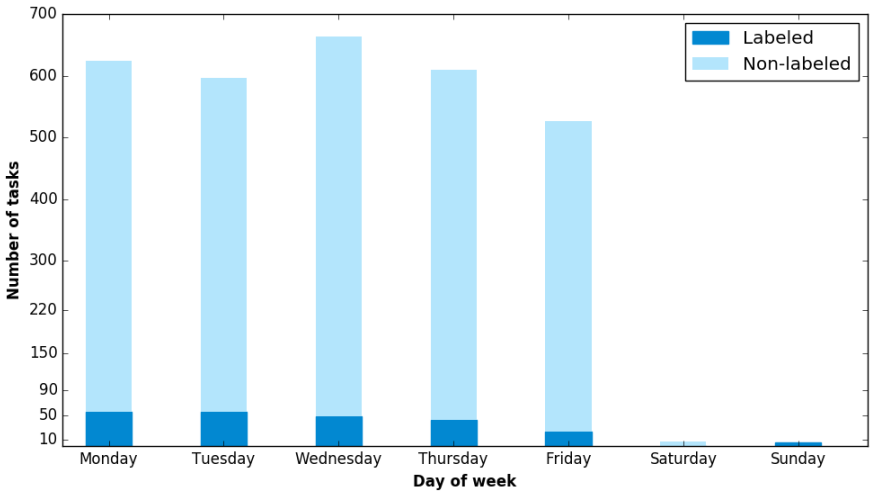

Figure 11: All collected data distributed daily.

We further analyze the collected data with daily and hourly distributions of collected data and their average difficulties. In case we use two accents, the darker color signifies labeled tasks and the lighter non-labeled, whereas both together show the total number of data points. On a simple histogram (Figure 11) we show the ratio between labeled and non-labeled data per day, from Monday to Sunday. We can clearly see that almost no data is collected during weekends. This is mainly due to the standard working hours – from Monday to Friday – and the office hours selection feature in TaskyApp (disabled automatic sensing on weekends). The data collected on weekends is

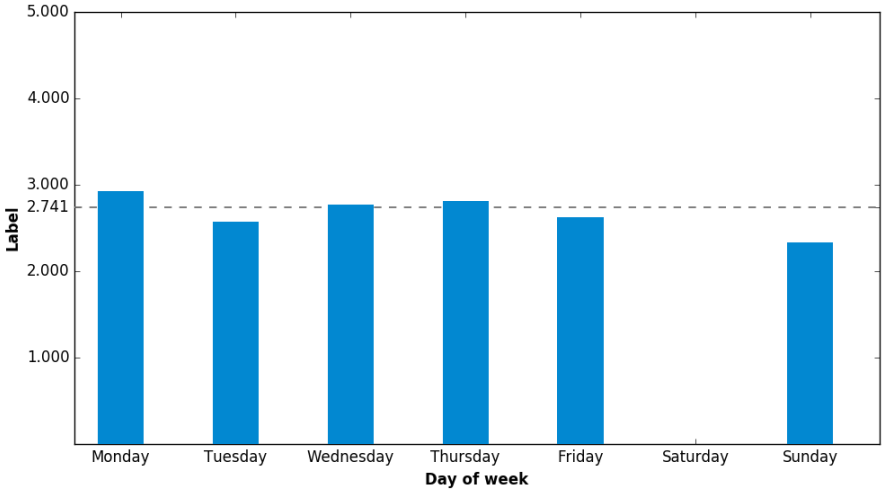

Figure 12: Average task difficulty distribution per day aggregated for all users.

either because of manually initiated sensing (Sunday) or by selecting weekends as office (working) days resulting in automatic sensing (Saturday). The figure shows us that the collected data is more or less evenly distributed over the weekdays. The number of labeled tasks differs from 24 on Fridays to 56 on Tuesdays. It looks like the users are not able to provide the same amount of labeled data on Fridays, where we see workload or tiredness (last working day of the week) as the main reasons. The number of non-labeled data points is fairly constant, it only differs due to the context switch detection feature in TaskyApp, from 503 collected on Fridays and 616 collected on Wednesdays, which may again indicate that users are more active at the start of the week.

Further, we analyze which day of the week is reported as the hardest (Figure 12). We encode tasks with numeric values from 1 (very easy) to 5 (very hard). The average task engagement level (dashed line) seems pretty constantly close to the mean value over the days, with Sunday being slightly easier and Monday slightly harder compared to the others.

We also investigate what time of the day we get the most data at (Fig-

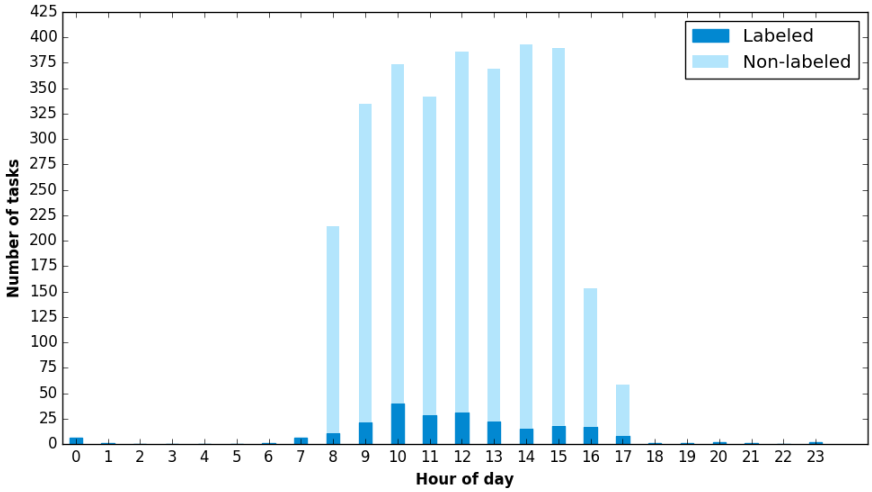

Figure 13: All collected data distributed per hour of the day.

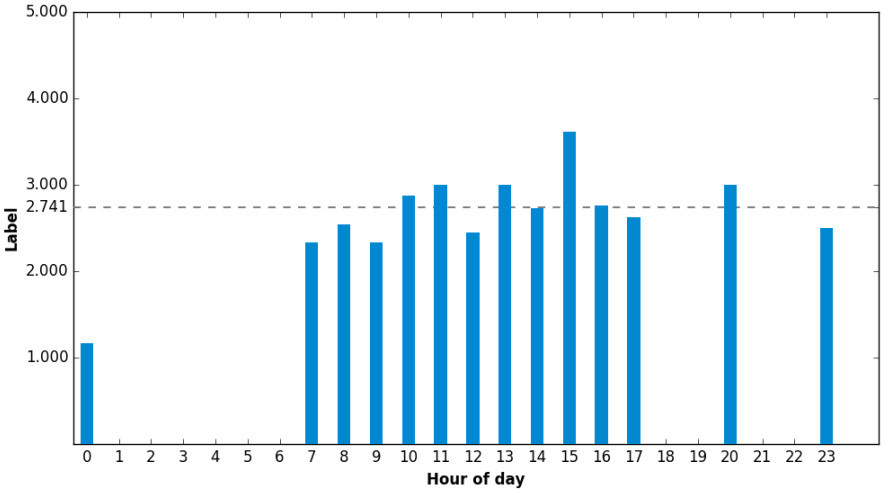

Figure 14: Average task label distribution per hour of the day aggregated for all users.

ure 13). Due to the office hours feature in TaskyApp, we collect most of the data during the default period from 8:00 to 16:00. The highest number of labeled data points, 40, is provided between 10:00 and 11:00. The number

drops to a single sensing until 19:00. The number of non-labeled tasks is constant between 9:00 and 17:00, meaning that most of the users work in the office during that time, whereas some might have changed the end of office hours to 18:00.

Knowing the amount of data collected over the day we also investigate the average task difficulty of reported labels per hour (Figure 14). The dashed line denotes the average task difficulty of the data points. Hours, in which we collect only one labeled task (e.g. 18:00-19:00), are left out. It looks like tasks reported at the end of office hours are slightly more difficult than in the mornings, but just by looking at the hour of the day we cannot predict if a task is going to be difficult or not.

Chapter 6

Data analysis

After successful data collection, we need to mine the data for features and find the desired correlation between task engagement and smartphone usage. As we have shown in the Section 5.2 we have a fairly small dataset with one user standing out for the number of provided labeled tasks. This leads us to do machine learning first on the whole dataset and then also only for that particular user, to test if we can find a better link. In this chapter, we discuss both, data mining and machine learning approaches we use in order to find the desired link.

6.1 Feature extraction

For this data mining process, we use Python programming language, as it has good programming libraries for data manipulation and an API to connect to the server's MongoDB. We write a script to generate two ARFF files for WEKA machine learning toolbox, one for a single user and another for the whole dataset of collected labeled tasks. The features we extract are the same in both files. Those files are later used for task engagement modeling.

We extract simple numeric features like number of WiFi and Bluetooth devices nearby, a number of active calendar events, an hour of the day and most importantly – the task's label. More complex numeric features are

extracted out of raw data collected using the accelerometer, gyroscope, and microphone. Accelerometer and gyroscope both have three axes (x, y and z), enabling us to extract similar features. First, we calculate arithmetic means for each axis since they are easy to compute and give us the first insights of how actively the phone is used. Next, as both sensors would need to be calibrated before the first sensing to get the exact features of phone movements and rotations, we decide to extract mean intensity (6.1), combining data of all three axes into a single value.

$$I = \sum_{i=1}^{N} \frac{\sqrt{x_i^2 + y_i^2 + z_i^2}}{N} \tag{6.1}$$

Next, we extract average variance of the intensity (6.2) to get a notion of how far the data is spread around the mean value.

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (I_i - \tilde{I})^2 \tag{6.2}$$

We also extract mean intensity's crossing values (6.3) to get a degree of phone's level of movement during the ten-second sensing window. We denote it with *mcr* and define it similarly to zero-crossings rate. The higher the output number is, the more the phone was moving (accelerometer data) or rotating (gyroscope data).

$$mcr = \sum_{i=2}^{N} \mathbb{R}_{<0}[sgn(I_i - \tilde{I}) sgn(I_{i-1} - \tilde{I})] \tag{6.3}$$

The mean-crossing rate is used also with microphone data. The data has a one-dimensional array of sound amplitudes, therefore we do not calculate intensities.

All the described features are from the time domain, thus we transform data to the frequency domain using Fast-Fourier Transform (FFT) and extract additional features for speech and activity recognition [27]. We calculate

mean of frequency power spectrum (6.4) and spectral entropy (6.8) for each axis and the microphone amplitude.

$$\widetilde{PS} = \frac{1}{N}\sum(|FFT(x)|)^2 \tag{6.4}$$

$$\tag{6.5}$$

To calculate spectral entropy, we again use FFT to calculate Power Spectral Density ($PSD(f)$). Then, we normalize that signal to get $PSD_n(f)$ and compute its Shannon entropy to get the feature's value (6.8). The entropy gives us a sense of unpredictability characteristics – information contained – in the power spectrum.

$$PSD(f) = |FFT(x)|^2 \tag{6.6}$$

$$PSD_n(f) = \frac{PSD(f)}{\sqrt[2]{|PSD(f)|^2}} \tag{6.7}$$

$$E_{PSD} = -\sum PSD_n(f) log_2[PSD_n(f)] \tag{6.8}$$

Apart from numeric values we extract also nominal values, such as: is screen turned on, is phone being charged and activity detected by Google Activity Recognition API.

6.2 Task engagement modeling

The ARFF file computed on the whole dataset in the data mining process is used to build machine learning models in this section. We mainly rely on open source machine learning software WEKA. It contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization [25]. In this section, we first focus on linear regression and later on classification models to test whether the collected data is (and how) linked with users' task engagement.

6.2.1 Linear regression

We first attempt a fine-grain inference of task engagement using linear regression. In linear regression we look for data correlations between dependent variable – task label – and independent variables, the rest of extracted features. That is done by fitting a linear line through cloud of our data points. Task label is encoded as a numeric value from 1 to 5 ("very easy" to "very hard").

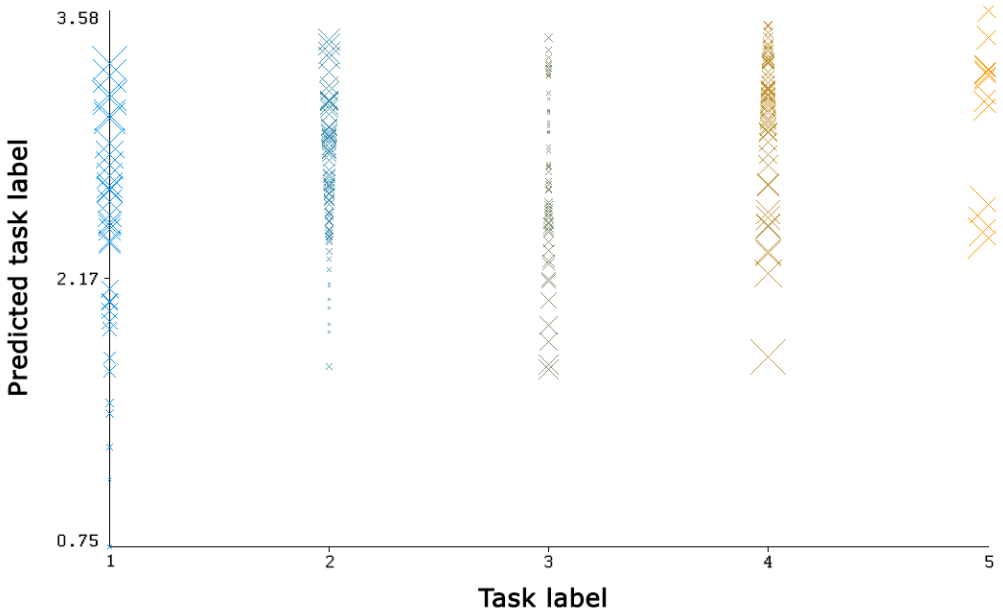

Figure 15: Error visualization of linear regression run on the whole dataset. Y-axis presents predicted and X-axis actual task labels. The bigger the appropriate X sign, the bigger the error.

We run the regression on only some of the extracted features. We choose them with first using WEKA's built-in attribute selection algorithm and then experimentally add and remove some to pick the features that together yield the highest R^2 parameter. We get the first confirmation that there is an existing link between task engagement and smartphone sensor readings. Some desired regression information reporting is not available in WEKA,

Variable	**Coefficient** (Std. Err.)	**t** (Sig.)
Accel. Y-axis mean	-.038	-1.84
	(.02)	(.068)
Accel. Z-axis mean	.026	1.43
	(.02)	(.153)
Accel. mean intensity	-.711	-3.04**
	(.23)	(.003)
Gyro. mean intensity crossing rate	.003	4.06**
	(.00)	(.000)
Gyro. intensity variance	.200	1.24
	(.16)	(.217
Hour of day	.067	3.49**
	(.02)	(.001)
(Regression Constant)	8.385	3.63
	(2.31)	(.000)

N	232
R^2	0.19
F	8.64 (p=.000)**

Table 3: Linear regression model built on top of some extracted features from the collected dataset.

thus we use statistical analysis program PSPP [8] to get all desired coefficients (Table 3). Statistically, there is a significant correlation with task engagement (F = 8.64 and p =.000). However, the R^2 value is fairly small, meaning that the model explains the variability of data only in 19% – there may be other factors that also affect task engagement. Further, the model predicts tasks with a mean absolute error of 0.845, which may be acceptable for

many practical purposes. Missing prediction by less than a task difficulty on the five-level scale is not critical (e.g. the biggest mistake here tends to be predicting a "neither easy nor hard" task as either "pretty easy" or "pretty hard" and vice versa). The error distribution scatter plot (Figure 15) is a bit more concerning. X-axis presents actual task labels on the scale from 1 to 5, whereas Y-axis presents values of predicted labels from 0.75 to 3.87. The model never predicts "very hard" task and only a few "very easy" tasks. The most of the predicted labels are around the mean value, consequently, the absolute error is reasonably low.

Further, we are interested in which variables are most correlated with inferring task engagement. Regression variables' coefficients indicate that when the phone moves a lot (accelerometer features' coefficients are negative, especially mean intensity) the task engagement label value leans towards easier tasks. This result is intuitive – as we are limited to the office settings, we expect that the phone is mostly on the table or in the user's pocket while the user is working. Movements of the phone indicate that the user has some free time to interact with the phone (e.g. checking phone calls, social media, etc.). Gyroscope's intensity variance has the second most significant coefficient, but much less strong than the accelerometer's mean intensity. Phone's rotations most likely result in harder tasks (e.g. the phone is flat on a desk and the user plays with it). A small role in inferring user's task engagement plays also the time of the day. Later in the day users tend to report harder tasks. The regression model consists of six variables, where three of them have significance lower than the common alpha level of 5% (denoted by **). This means that there is only a very small probability we got those variable values by chance.

6.2.2 Classification

Although the above regression analysis points out to a link between task engagement levels and sensor data readings, a fine-grain distinction among engagement levels is difficult. Therefore, we decide to re-encode our labels

into only two classes: *"easy"* and *"difficult"*. Previously labeled "very easy" and "pretty easy" tasks are now labeled as "easy" and all the other label values are labeled as "difficult". This classification gives us a balanced set of data – 107 labeled as "easy" and 125 as "difficult".

We test our data using informative features, reported by linear regression, on three different classifiers in WEKA:

- **ZeroR**
- **Naive Bayes**
- **Random Forest**

We then evaluate each classifier by performing a ten-fold cross-validation. In a 10-fold cross-validation, the data is randomly divided into ten equal-sized pieces. Each piece is used as the test set with training done on remaining 90% of the data. The test results are then averaged over the ten cases.

In discussing each classifier's result, we pay the most attention to the overall prediction accuracy and Type 1 errors – predicting difficult tasks as easy. For practical purposes, such mistakes are expensive. A messaging app based on a classifier with a high percentage of Type 1 errors would predict that a highly engaged user is free for an interruption, and would disturb the user at an inappropriate moment. Moreover, a health care human resource management system would predict that a very busy doctor is not engaged in a difficult task and would assign her to an urgent intervention. The other type of errors, Type 2, are less critical. The messaging app would not disturb the user, at most, it would annoy her for showing the message with delay.

ZeroR

First, we run majority class classifier – ZeroR. It is a very simple classifier that always predicts the majority class in a dataset. We use it in our study as a baseline to evaluate the performance of other classifiers. The majority class of our dataset is "difficult" (Table 4). We denote predicted values with

an apostrophe (’). Therefore, ZeroR classifier is successful in 53,9%, resulting in no Type 1 errors and 46,1% of Type 2.

easy’	**difficult’**	
0 (0%)	107 (46,1%)	**easy**
0 (0%)	125 (53,9%)	**difficult**

Table 4: Confusion matrix gained by running the baseline classifier – ZeroR.

Naive Bayes

Naive Bayes is a probabilistic classifier based on applying Bayes’ theorem with strong (naive) independence assumptions between the features. Rish demonstrates that naive Bayes works best in two cases: completely independent features and functionally dependent features [31].

easy’	**difficult’**	
55 (23,7%)	52 (22,4%)	**easy**
32 (13,8%)	93 (40,1%)	**difficult**

Table 5: Confusion matrix of Naive Bayes classifier.

Off the shelf naive Bayes classifier in WEKA provides us with the prediction accuracy of 63,8%, which is superior to the baseline for 9,9%. The confusion matrix (Table 5) shows that the classifier fails in correctly labeling difficult tasks in 13,8% of predictions, which we deem as acceptable. On the other hand, it has 22,4% of Type 2 errors (more than 20% less than the baseline).

Random forest

We then try random forest classifier, which is a divide-and-conquer approach used to improve performance. The main principle behind is that a group of

"weak learners" (decision trees) are joined to form a "strong learner" (forest). At first the algorithm builds a few decision trees, each based on around 66% of randomly selected subset of the same data. Each tree, especially deep, is usually overfitted to data, due to low bias, but high variance properties. After a large number of trees are generated, they vote for the most popular class [5].

easy'	**difficult'**	
66 (28,4%)	41 (17,7%)	**easy**
57 (24,6%)	68 (29,3%)	**difficult**

Table 6: Random forest's confusion matrix.

Random forest implementation in WEKA results in 57,8% correctly predicted tasks (Table 6), which is again better than the baseline prediction, but inferior to the prediction of naive Bayes. Also, the error distribution indicates that this classifier is not as practical as naive Bayes is. The percentage of critical error is higher by almost eleven percent, whereas, non-critical error percentage is lower by nearly five percent.

6.2.3 Individual task engagement modeling

Task engagement modeling indicates at a (very weak) link between smartphone sensor readings and user's task engagement level. There is a potential for improved results with an individual user's data analysis [3, 26]. The collected data is of eight different users, each having their own habits on where to put a phone during sensing and how actively they use it during office hours among others. Thus we decide to study the data provided by a user that stands out with the most reported labeled tasks. In this section, we load the other ARFF file in WEKA and run the same machine learning algorithms on 83 records of user 1 (Table 2) to test if we can achieve more accurate results.

Linear regression

We again use the same approach to choose the most informative features and end up building a linear regression model using five different features (Table 7). The individual's regression model explains data variability much better (R^2 is 36,4%) and has a lower mean absolute error. Looking at coefficients' significance levels, only hour of day, accelerometer intensity crossing rate and intensity variance have the significance lower than 5% (denoted **).

Variable	**Coefficient** (Std. Err.)	**t** (Sig.)
Accel. mean intensity	-1.64	.97
	(-1.69)	(.095)
Accel. mean intensity crossing rate	0.01	2.44**
	(.00)	(.017)
Accel. intensity variance	3.25	2.85**
	(1.14)	(.006)
Gyro. Z-axis mean	15.43	1.81
	(8.53)	(.074)
Hour of day	.11	3.60**
	(.03)	(.001)
(Regression Constant)	15.925	1.57
	(10.14)	(.120)
N	83	
R^2	0.364	
F	8.81 (p=.000)**	

Table 7: Linear regression model applied to the top provider's data features. Three variables are significant.

The latter feature affects the end result the most, with a coefficient of 3.25, meaning that movements indicate at harder tasks. This is contrary to the general model in the previous section. A possible reason could be different working habits or a different type of work of that particular participant compared to the others. Gyroscope Z-axis readings affect the end result in the same way as the general model does – phone rotations are related to harder tasks. The same goes with the hour of day feature, the latter in the day the harder the reported task.

Classification

Again, we first define a baseline using the built-in WEKA's majority class classifier – ZeroR. We now get 40 labeled as "easy" and 43 as "difficult" tasks, thus the baseline (Table 8) is now at 51,8%, lower by 2,1% compared to the general model. All classifiers are ran on the same features and are chosen from the individual's linear regression model (Table 7). We use the same built-in classifiers in WEKA as for the general model and evaluate them using the ten-fold cross-validation method. Then, we run naive Bayes

easy'	**difficult'**	
0 (0%)	40 (48,2%)	**easy**
0 (0%)	43 (51,8%)	**difficult**

Table 8: Top provider's confusion matrix gained by running ZeroR, baseline, classifier.

classifier to reach 62,7% prediction accuracy (Table 9). It is again higher compared to our baseline, in this case by 10,9%, but lower by 1,1% than naive Bayes model on the whole dataset. At first glance, this is inferior, but comparing the differences between corresponding baselines, this model achieves a higher accuracy by one percent. Also, it has a significantly lower rate of critical errors (9%), making this model more practical and safer to use. On the other hand, Type 2 error rate is higher by 10,1%.

easy'	**difficult'**	
13 (15,7%)	27 (32,5%)	**easy**
4 (4,8%)	39 (47,0%)	**difficult**

Table 9: Confusion matrix of Naive Bayes classifier for the single participant.

Next, we use Random forest algorithm and achieve 61,4% accuracy (Table 10). This is, again, higher than the baseline by nearly 10%. Naive Bayes is not only more accurate, but also has a lower critical error rate by almost ten percent. Comparing the results of Random forest algorithm ran on the data of the single user with the general model, we achieve 3,5% higher accuracy for the personalized classifier.

easy'	**difficult'**	
20 (24,1%)	20 (24,1%)	**easy**
12 (14,5%)	31 (37,3%)	**difficult**

Table 10: Random forest's confusion matrix for the single participant.

Chapter 7

Limitations and future work

Smartphones were not originally envisioned for inferring cognitive states, nor are used in a manner that makes such an inference straightforward. Therefore, we limit our study to users working in offices, as such environments are rich in task dynamics, with many different tasks of various difficulties. We manage to find only 10 suitable participants willing to participate and collect only 232 labeled sensor readings. Also, we depend on subjectively reported task labels and assume that all the collected data is labeled correctly. Instead, we could use some smart filters to exclude inappropriately reported data points from task engagement modeling (e.g. detect that a particular sensor reading was sensed while the user was riding a bike). Moreover, sensor calibration in the app would enable us to extract more quality features, e.g. exact rotation of the phone. The calibration, of accelerometer and gyroscope in particular, could improve the end result, but would require higher user engagement.

Certain avenues have not been explored in our research and could improve the end result. Aside from 232 labeled data points, we have also collected 2802 non-labeled sensor readings that have not been analyzed yet. Therefore, semi-supervised learning could be applied to build more accurate classifiers [41]. Besides, there is an unlimited number of features available to be extracted from the collected data and it is possible that we did not

extract the most relevant ones. Due to the small dataset collected, we did not extract features from location data, which could be used at least for personalized classifiers. Since we identify the phone movements as the most informative features, some features proposed by Lester et al. for their activity recognition system [22] could potentially boost the inference of task engagement.

The importance of features coming from accelerometer and gyroscope point us towards our next step – task engagement inference using automated sensing on wearable devices. Smartphones are usually held somewhere close to the user, but are not always physically involved in the ongoing tasks (e.g. the phone is in a bag while the user is writing an email). Hence, the user's activity recognition during a task could be much better modeled using on-body sensor devices, such as smartwatches and fitness tracker wristbands. Not only they are worn all the time, but also offer alternative types of sensors. Affordable wristbands offer heart rate monitoring, skin temperature, and step counters among others, whereas more sophisticated devices equip us with detection of electrodermal activity and enable stress inference [9]. We have already started upgrading TaskyApp with the integration of two wristbands – AngelSensor and Xiaomi Mi Band. The main advantages of AngelSensor are its open communication protocols, API/SDK, and sensor data streams [23]. It has a wide range of sensors, where, in the integration with TaskyApp, we utilize heart rate and skin temperature readings at the moment, but we also plan to read raw accelerometer data in the future. However, the wristband is not widely used and has a fairly high price tag at $99 USD, hence we also consider Xiaomi Mi Band 1S [40]. At around five times lower price, Mi Band is a very affordable device and has a larger community of users, but does not have a publicly available API and has only a limited range of sensors. Its accelerometer cannot be read via smartphones, thus we have only managed to use its heart rate monitor by using a reverse engineered API.

Chapter 8

Conclusion

Most modern mobile applications do not pay attention to users' availability, thus end up in their annoyance and failure of reaching the common goal – user engagement. This may reflect in a user ignoring the app's request for input (e.g. a notification or an ad), in a lower monetization of the app, higher usage of system resources, bad reviews on stores, app uninstalls or other unwanted consequences. Inferring task engagement could be beneficial in particular for messaging apps (to defer an unimportant message), news apps (users tend to read the news when bored [30]) or personal assistance apps like Google Now or Siri to update nonessential content when less engaged in a task, thus saving battery and mobile data usage. Elsewhere, being able to infer task engagement could largely benefit human resource management systems, helping distribute workload and reduce costs.

In this work, we explore task engagement detection through mobile sensing in office settings. We develop TaskyApp, a mobile application to collect data from various built-in sensors shipped in modern smartphones. We run the app among volunteers and collect 232 labeled sensor readings of 8 different users. We use the collected labeled data to extract features and build machine learning models. First, we use linear regression to confirm that the link between user's task engagement and sensor readings exists and identify movement and time as the most informative features. Afterward, we build

classification models and predict task engagement with an accuracy of up to 63,8% using off the shelf classifiers. We are further interested in whether we can build a better model using individual's data. We repeat the modeling process on 83 data points of a single user's data and reach an even higher accuracy compared to the baseline. Again, we confirm similar informative features as for the whole dataset.

Conducting the study lead us to some more and some less anticipated results. During the study, we discovered how different functionalities in the app engage users into using the app and the effect of our UI decisions on its usage. We further discover that it is very challenging to engage users in using the mobile application and provide sufficient amount of labeled data. We also find out that collected data is highly personalized, probably due to different working habits and environments among users. Linear regression models confirm that task engagement mostly depends on accelerometer and gyroscope features as well as the time of the day. Surprisingly, features like the number of nearby Bluetooth and WiFi devices, features of accelerometer and gyroscope in the frequency domain, ambient sound, calendar events, charging status and volume settings do not improve our inference model. Although the prediction accuracy at this point is still fairly low, some apps could already benefit from the results, e.g. news apps for determining the time of updates to save battery and mobile data. Nevertheless, with the expected growth of sensor devices in our everyday environment the potential for automatic task engagement is yet to grow.

Bibliography

[1] Piotr D. Adamczyk and Brian P. Bailey. If not now, when?: The effects of interruption at different moments within task execution. In *ACM CHI'04*, Vienna, Austria, April 2004.

[2] John R. Anderson. *How can the human mind occur in the physical universe?* Oxford University Press, USA, July 2007.

[3] James Bo Begole, Nicholas E. Matsakis, and John C. Tang. Lilsys: sensing unavailability. In *CSCW'04*, Chicago, IL, USA, November 2004.

[4] Jelmer P. Borst, Niels A. Taatgen, and Hedderik van Rijn. The problem state: A cognitive bottleneck in multitasking. *Journal of Experimental Psychology: Learning, memory, and cognition*, 36(2):363–382, March 2010.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, October 2001.

[6] Thomas H. Davenport and John C. Beck. *The Attention Economy: Understanding the New Currency of Business*. Harvard Business Review Press, September 2002.

[7] James Fogarty, Scott E. Hudson, Christopher G. Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C. Lee, and Jie Yang. Predicting Human Interruptibility with Sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(1):119–146, March 2005.

[8] Inc. Free Software Foundation. PSPP - GNU Project - Free Software Foundation. Available: https://www.gnu.org/software/pspp/, 2016. (Acquired: 26.08.2016).

[9] Martin Gjoreski, Hristijan Gjoreski, Mitja Luštrek, and Matjaž Gams. Continuous Stress Detection Using a Wrist Device – In Laboratory and Real Life. In *Mental Health and Well-being workshop with ACM UbiComp'16*. ACM, September 2016.

[10] Victor M. González and Gloria Mark. Constant, constant, multi-tasking craziness: managing multiple working spheres. In *ACM CHI'04*, Vienna, Austria, April 2004.

[11] Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work?–a literature review of empirical studies on gamification. In *2014 47th Hawaii International Conference on System Sciences*, Waikoloa, HI, USA, Januar 2014.

[12] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In *ICMI'03*, Vancouver, Canada, November 2003.

[13] Google Inc. Activities | Android Developers. Available: https://developer.android.com/guide/components/activities.html, 2016. (Acquired: 17.08.2016).

[14] Google Inc. Dashboards | Android Developers. Available: https://developer.android.com/about/dashboards/index.html, 2016. (Acquired: 21.01.2016).

[15] Google Inc. Material design - Google design guidelines. Available: https://material.google.com/, 2016. (Acquired: 28.07.2016).

[16] MongoDB Inc. The MongoDB 3.2 Manual — MongoDB Manual 3.2. Available: https://docs.mongodb.com/manual/, 2016. (Acquired: 03.08.2016).

[17] Statista Inc. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016. Available: `http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/`, 2016. (Acquired: 11.08.2016).

[18] Statista Inc. Number of smartphone users worldwide from 2014 to 2019 (in millions). Available: `http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`, 2016. (Acquired: 08.09.2016).

[19] Twitter Inc. Fabric - Twitter's Mobile Development Platform. Available: `https://get.fabric.io/`, 2016. (Acquired: 26.09.2016).

[20] Shamsi T. Iqbal, Xianjun Sam Zheng, and Brian P. Bailey. Task-evoked pupillary response to mental workload in human-computer interaction. In *ACM CHI'04 (extended abstracts)*, Vienna, Austria, April 2004.

[21] Neal Lathia, Kiran Rachuri, Cecilia Mascolo, and George Roussos. Open source smartphone libraries for computational social science. In *MCSS'13*, Zürich, Switzerland, September 2013.

[22] Jonathan Lester, Tanzeem Choudhury, and Gaetano Borriello. A practical approach to recognizing physical activities. In *IEEE PerCom'06*, Pisa, Italy, March 2006.

[23] Seraphim Sense Ltd. Angel Sensor – Open Mobile Health Wearable | The future of health and well being. Available: `http://angelsensor.com/`, 2016. (Acquired: 01.09.2016).

[24] Heather L. O'Brien and Elaine G. Toms. What is user engagement? A conceptual framework for defining user engagement with technology. *Journal of the American Society for Information Science and Technology*, 59(6):938–955, April 2008.

[25] The University of Waikato. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. Available: `http://www.cs.waikato.ac.nz/ml/weka/`, 2016. (Acquired: 16.08.2016).

[26] Veljko Pejović and Mirco Musolesi. InterruptMe: Designing intelligent prompting mechanisms for pervasive applications. In *ACM UbiComp'14*, Seattle, WA, USA, September 2014.

[27] Veljko Pejović and Mirco Musolesi. Anticipatory mobile computing: A survey of the state of the art and research challenges. *ACM Computing Surveys (CSUR)*, 47(3):47, April 2015.

[28] Veljko Pejović, Mirco Musolesi, and Abhinav Mehrotra. Investigating The Role of Task Engagement in Mobile Interruptibility. In *Smartention, Please! Intelligent Attention Management on Mobile Devices Workshop (with MobileHCI'15)*, Copenhagen, Denmark, August 2015.

[29] Martin Pielot, Rodrigo de Oliveira, Haewoon Kwak, and Nuria Oliver. Didn't you see my message? predicting attentiveness to mobile instant messages. In *ACM CHI'14*, Toronto, ON, Canada, April 2014.

[30] Martin Pielot, Tilman Dingler, Jose San Pedro, and Nuria Oliver. When attention is not scarce-detecting boredom from mobile phone usage. In *ACM UbiComp'15*, Osaka, Japan, August 2015.

[31] Irina Rish. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on Empirical Methods in AI*, Seattle, USA, August 2001.

[32] Dario D. Salvucci and Niels A. Taatgen. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review*, 115(1):101–130, Januar 2008.

[33] Dario D. Salvucci and Niels A. Taatgen. *The multitasking mind*. Oxford University Press, September 2010.

[34] Gašper Urh. TaskyApp - Android Apps on Google Play. Available: `https://play.google.com/store/apps/details?id=si.uni_lj.fri.taskyapp`, 2016. (Acquired: 06.08.2016).

[35] Gašper Urh and Veljko Pejović. Taskyapp: Inferring task engagement via smartphone sensing. In *Ubittention workshop with ACM UbiComp'16*, Heidelberg, Germany, September 2016.

[36] Gašper Urh and Veljko Pejović. TaskyApp | FRI research project. Available: `http://193.2.72.121/`, 2016. (Acquired: 06.08.2016).

[37] Gašper Urh and Veljko Pejović. urgas9/TaskyApp. Available: `https://github.com/urgas9/TaskyApp`, 2016. (Acquired: 27.09.2016).

[38] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, September 1991.

[39] Jason Wiese, T. Scott Saponas, and A.J. Bernheim Brush. Phoneprioception: Enabling mobile phones to infer where they are kept. In *ACM CHI'13*, New York, NY, USA, April 2013.

[40] Xiaomi. Mi Band - Mi Global Home. Available: `http://www.mi.com/en/miband/`, 2016. (Acquired: 16.09.2016).

[41] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.